1989-07

A Reply to Djikstra's Paper

Hamming, Richard W.

Monterey, California: Naval Postgraduate School

A REPLY TO DIJKSTRA'S PAPER

R. W. Hamming
Naval Postgraduate School
Monterey, CA 93943

Perhaps it is best to begin by recalling that long ago Dijkstra led a crusade for the *total* abolition of the GOTO instruction. Currently it is widely believed that the GOTO instruction is used much too often but that it also has its place in programming.

The present paper is another example of Moses laying down the law to us sinners in programming. As before it is both very right and very wrong.

Unfortunately in his paper there is much "sound and fury" and non sequiturs, the extensive use of color words, a gratuitous swipe at the military, and often little content beyond his assertions; you can detect the extent of this if you try to rewrite his paper, as I did, in simple language and clear reasoning.

Just as in the GOTO paper, there is much truth in this paper. I agree wholeheartedly that we should replace the word "bug" with the word "error" and suppress all anthropomorphic words as being misleading to the beginners.

The major trouble is, I think, that Dijkstra believes that programming should resemble mathematics and believes that mathematics is what one is taught à la Euclid. One first lays down postulates, makes a few appropriate definitions, states theorems, and then proves them - after all that is how mathematics is typically taught. He refuses to recognize that often the postulates follow from the theorems, as do many of the definitions, and often the theorems follow from the proofs we are able to generate - they are then called "proof driven theorems." Further more, Dijkstra in his sober moments well knows that human proofs in mathematics are unreliable, that many famous proofs have been repeatly "patched up" by subsequent generations, hence even if we tried to use his idea that programs should be "proved" by humans before they are run, the proofs are fallible, and in any case are merely paper problems run on a paper machine. It is this that I believe is *behind* much of the paper.

One of Dijkstra' major points is that the rapid growth of computers represents a unique change of magnitude, so large that no one can comprehend it; but large changes are more common than he thinks, for example the bandwidth available for signalling. The unique features he attributes to computer science occur in other fields of human activity: (1) particle accelerators have similarly increased in size and power consumption, (2) the complexity of the telephone system of interconnected central offices was around long before the first of the electronic computers and is still probably more complex than any computer (3) the claim of unique vulnerability to a single error is certainly shared by our common languages.

Dijkstra excoriates "software engineering" by deliberately comparing it to his conception of mathematics rather than to, say, "effective writing" which I feel is a far better analogy. I also doubt that it is always wise to equate a program to a mathematical

formula as he does.

Dijkstra uses the well known example of trying to cover with dominoes a checker board, without the diagonal corners, to illustrate the value of the mathematical, analytical approach and concludes immediately that *all* programs will similarly benefit - the reasoning is fallacious, and from him shocking!

Dijkstra flatly asserts that he knows "reality," and his opponents do not, but I put about as much faith in this as in the statement, "I am Napoleon." Indeed, in my opinion, he comes to grief simply because his "reality" is so far from most other people's, which he admits.

Dijkstra objects to measuring programming by lines of code, conveniently forgetting that authors are often paid by the word. In both cases it is ridiculous at times to do so, but he offers no other *practical* measure of programming (or writing) effort.

Dijkstra willfully misunderstands "software maintenance" pretending that it means repairing parts that have failed, rather than what everyone else understands, mainly altering the current software to meet changing needs and environments; hence, contrary to his sneer, software that is not maintained is apt to be of much less value to the user than software that is.

Dijkstra seems to identify programming languages with "imperative languages" and deigns not to notice "object oriented" and "functional" programming to mention two other approaches - after all he is Moses and he knows what programming is.

Returning to the main theme of this reply, his desire to map software onto his conception of mathematics is foolish. His idea that a program be "proved" to be correct before running it applies, as noted before, to a paper program on a paper machine, and not to reality. When you have to make a compiler for a new mathematically defined language then this approach is both very reasonable and valuable, but in many, if not most, engineering cases where the design criteria arise from what you are able to do, this approach simply does not work very well (as can be seen from the government procurement policy in action). Even more than in mathematics, in engineering there is a "give and take" between the design proposal and what can be done in current practice; hence his desire to start with an exact description for the program that is to be written works mainly in his "reality".

Anyone who reads the above objections to mean that what Dijkstra writes can be safely ignored is a fool. I have documented some of the errors that his extremism has produced, but there is much truth in his paper. Apparently reformers must often be extreme in what they say and do if they are achieve a reformation. Read with charity Dijkstra's paper is a valuable contribution; Moses has indeed lead us a bit further out of our wilderness.